Department of Computer Engineering, METU

CENG 140

Section 1 & 2

Spring 2018-2019

THE 1

Görkem Özer
gorkem@ceng.metu.edu.tr

Due date: 03.04.2019 Wednesday, 23:59

1 Overview

You are working for a large shipping company. Many companies work with your company to transport their goods to overseas. These companies are their customers.

The shipping company has a problem nowadays: it runs out of ships with ideal capacity when needed. Sometimes ships move with overcapacity, sometimes capacity is wasted and ships go empty.

This is where your job begins. They hired you to solve their optimization problem. Since the shipping company wants to serve more customers, they want you to minimize the number of ships spared for each customer. When the company serves multiple customers, reputation of the company will increase and it will dominate the market more.

2 The Fleet

- The fleet is not stationary, many ships come and go. Ships at harbour are in different types. Each one of them carries different amount of load. The company representative told you that there are **at most 10 distinct ship types** in the harbour at any time. You can also assume that ships with 1 unit capacity are always ready for shipment in the harbour.
- At any time, multiple customers can ask for company's service. When a customer comes, a captain among available captains in the harbour is assigned to the shipment. **At most 10 customers** can come and ask for company's service. The company representative also told you that they can accept **maximum amount of load of 1000 units** at any time for any customer.
- Above upper bound restrictions are added in implementation template files. Check macros inside them.
- **NOTICE:** Only important thing about serving customers is **sparing minimum number of ships** to them and having **as much satisfied customer as possible.** After you find how many ships a customer requires, you need to assign captains among available captains accordingly. The company's captains are highly-qualified, they can lead any type of ship they want, ship type is not an issue for them. **You are NOT responsible for finding which types of ships should be spared for each customer in this assignment.**

3 Tasks

1. Read input by implementing **read_input()** function in **the1_read_input.c** file. See **Regulations section** for input details. **(10 points)**
2. Calculate minimum number of ships spared for a customer in *"recursive"* manner.
   Implement **min_ships_required_recursive()** function in **the1_recursive.c** file. **(35 points)**
3. Calculate minimum number of ships spared for a customer in *"iterative"* manner.
   Implement **min_ships_required_iterative()** function in **the1_iterative.c** file. **(40 points)**

4. Find out how many customers the company can serve. Implement **serve_customers()** function in **the1_serve_cust.c** file. Return the number of customers with their requirements satisfied. If number of available captains in the harbour **is NOT enough** for any customer's required ships, **return 0**. Consider the notice in **The Fleet section.**
HINT: To serve as many customers as possible, you may use **sort_integer_array()** function in **the1_serve_cust.c**. **(15 points)**

4 Regulations

- **Input-Output format:** Input-output formats are given below. **Output format is prepared for you in test cases (in the1.c).** While evaluating your assignments, different main() function implementation will be used for more testing.

  Actually this is why you will implement this assignment in different files rather than a single file. With this way, your function implementations can be evaluated separately and easily plugged in and plugged out to the main code without touching your function implementations.

  Input Format:

```
<available_captain_count_in_the_harbour>
<ship_type_count>
<ship_type_1> <ship_type_2> ... <ship_type_n>
<customer_count>
<customer_requirement_1> <customer_requirement_2> ... <customer_requirement_n>
```

  Output Format (for testing purposes):

```
<min_ships_required_for_customer_0> ships required for customer 0
<min_ships_required_for_customer_1> ships required for customer 1
...
<min_ships_required_for_customer_n-1> ships required for customer n-1
<number_of_satisfied_customers>
```

  ASSUMPTIONS that you can make about inputs:

  - Ship types will be given to you in ascending order.
  - There will be no repetitions in ship types.
  - There will be always 1-unit-ships among ship types in the input.
  - With above assumptions, ship type row can be:
    1 3 4 7
    1 3 7
    1 2 6 10
    1 2 5 8
    1 x y z ...

  Example 1:
  There are 4 available captains and 4 different types of ships in the harbour. Types of these ships are ships with 1 unit, 5 units, 6 units and 9 units of capacity. There are currently 2 customers contacted with the company for its services. First customer (customer 0) wants to transport 40 units of goods, second customer (customer 1) wants to transport 30 units of goods.

  Result 1:
  Customer 0 = 40 units (9+9+9+6+6+1) needs 6 ships
  Customer 1 = 30 units (9+9+6+6) needs 4 ships
  With 4 available captains in the harbour, the company can serve to only customer 1.

  Input:

```
4
4
1 5 6 9
2
40 30
```

Output:

```
6 ships required for customer 0
4 ships required for customer 1
1
```

Example 2:
There are 12 available captains and 3 different types of ships in the harbour. Types of these ships are ships with 1 unit, 4 units and 7 units of capacity. There are currently 3 customers contacted with the company for its services. First customer (customer 0) wants to transport 35 units of goods, second customer (customer 1) wants to transport 45 units of goods, third customer (customer 2) wants to transport 25 units of goods.

Result 2:
Customer 0 = 35 units (7+7+7+7+7) needs 5 ships
Customer 1 = 45 units (7+7+7+7+7+7+1+1+1)
.....or (7+7+7+7+7+4+4+1+1)
.....or (7+7+7+7+4+4+4+4+1)
.....or (7+7+7+4+4+4+4+4+4) needs 9 ships
Customer 2 = 25 units (7+7+7+4) needs 4 ships
With 12 available captains in the harbour, the company can serve to customers 2 and customer 0 later.

Input:

```
12
3
1 4 7
3
35 45 25
```

Output:

```
5 ships required for customer 0
9 ships required for customer 1
4 ships required for customer 2
2
```

- **Programming Language:** C
- **Libraries and Language Elements:**
  You should not use any library other than *"stdio.h"*, *"math.h"* and *"limits.h"*. You can use conditional clauses (switch/if/else if/else), loops (for/while), arrays, 2D arrays. **You can NOT use any further elements beyond that (this is for students who repeat the course).** You can define your own helper functions, provided that you have implemented 4 functions given to you in **Tasks** section. Finally, please look at comments in the code for deep understanding about functions.
- **Compiling and running:**
  You should be able to compile your codes and run your program with following commands:

```
>_ gcc the1.c the1_read_input.c the1_recursive.c the1_iterative.c
    the1_serve_cust.c -ansi -pedantic-errors -Wall -lm -o the1
>_ ./the1
```

  If you are working with ineks or you are working on Ubuntu OS, you can feed your program with input files instead of typing inputs. This is a feature of Ubuntu OS and an equivalent of typing inputs. This way you will test your inputs faster:

```
>_ ./the1 < inp1.txt
>_ ./the1 < inp2.txt
```

- **Submission:**
  You will compress **the1_read_input.c, the1_recursive.c, the1_iterative.c, the1_serve_cust.c** implementations and submit compressed file as **e1234567_the1.zip** to the CengClass where 1234567 should be REPLACED with your OWN student id. Since submission is a part of the assignment, any error on submission file will be penalized. Late submission IS NOT allowed, it is not possible to extend the deadline and **please do not ask for any deadline extensions**.

- **Evaluation:** Your codes will be evaluated based on several input files including, but not limited to the test cases given to you as example. If your program gives correct outputs for all cases, you will get 100 points.

  Your *"recursive"* solution should work fast for small values. When values get larger, *"recursive"* function returns after some considerable time. This is the natural behaviour, do not worry. On the other hand, your *"iterative"* solution should work fast for both small and large values.

- **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations and will get 0. Sharing code between each other or using third party code is strictly forbidden. Even if you take a "part" of the code from somewhere/somebody else - this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar. So please do not think you can get away with by changing a code obtained from another source.